Étape 6 : Déplacement aléatoire.

Jamila Sam & Jean-Cédric Chappelier, 2018

Version : 1.1

But

Ajout d'aléatoire dans le déplacement des animaux.

Description

Le but de cette étape est de modifier l'algorithme de déplacement des animaux de sorte à ce qu'ils puissent faire de temps à autre des rotations aléatoires, car, à courir tout droit devant eux, ces animaux ont peu de chance de tomber sur des choses intéressantes, comme de la nourriture par exemple.

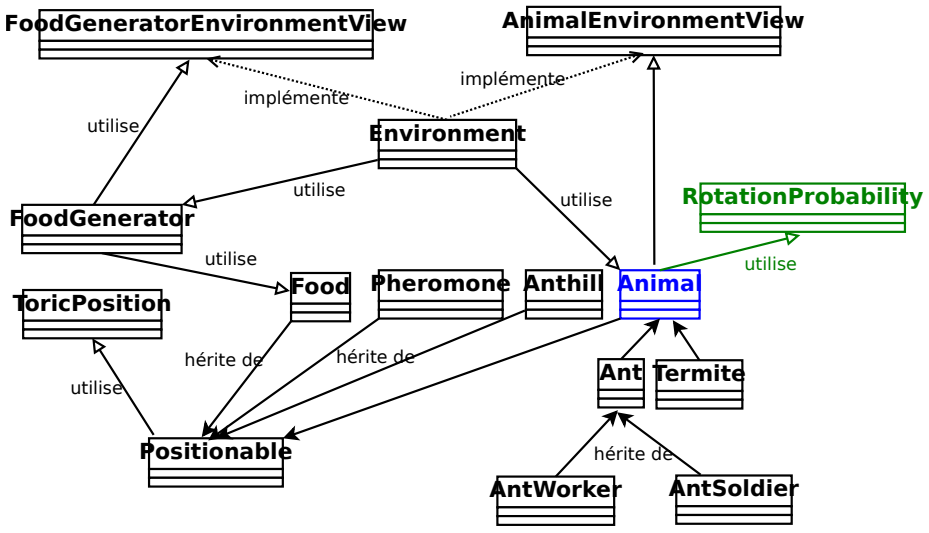

Figure 1: Architecture codée au terme de l'étape 6 (en noir les composants déjà codés, en vert les nouveaux composants, en bleu ceux retouchés)

Vous aurez à introduire une nouvelle classe `RotationProbability` permettant de gérer les changements de direction aléatoires et aurez à retoucher à la méthode de déplacement des animaux codée à l'étape précédente.

La figure 1 donne une vue d'ensemble de l'architecture à laquelle vous allez aboutir au terme de cette étape.

Algorithme

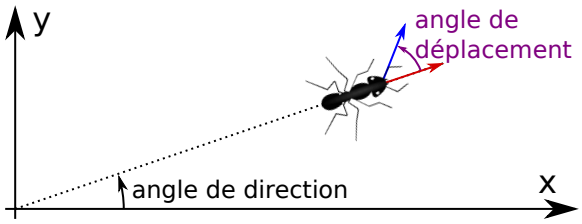

Figure 2: angle de déplacement des animaux.

Chaque animal se déplace par rapport à son orientation propre, c.-à-d. par rapport à sa direction actuelle (voir figure 2). Pour simuler les écarts de déplacement, on utilisera deux tableaux :

- un tableau d'angles de déplacement (donnés en degrés car plus parlant pour nous) ; par exemple :

  ```
  ( -180, -135, -90, -45, 0, 45, 90, 135, 180 )
  ```

- un tableau P_m de probabilités associées à chacun de ces angles ; par exemple :

  ```
  ( 0.00, 0.01, 0.09, 0.15, 0.50, 0.15, 0.09, 0.01, 0.00 )
  ```

  (l'ensemble de ces valeurs somme donc à 1).

A chaque cycle de simulation (ou presque), un angle de rotation qui va conditionner le changement de direction de l'animal sera tiré aléatoirement. L'animal aura alors une probabilité $P_m(n)$ de faire subir à sa direction une rotation donnée par l'angle à la position n dans le tableau des angles ; par exemple avec les valeurs ci-dessus, l'animal a une probabilité 0.01 de tourner de -135 degrés et 50% de chance de rester dans la même direction (angle 0 degrés).

Physiquement, P_m est l'ensemble des probabilités dites « *inertielles* » représentant le fait qu'un animal est un objet en mouvement dans le monde et ne change pas de direction d'un seul coup : il possède une certaine inertie. Cette probabilité est donc maximale lorsque l'animal ne change pas de direction (continue son chemin tout droit, angle 0) et nulle pour un retournement complet. Elle est de plus symétrique par rapport à la direction de l'animal.

Implémentation

La classe `RotationProbability`

Nous commencerons par définir un nouveau type permettant d'associer un tableau d'angles à un tableau de probabilités. La classe à créer pour cela s'appelle `RotationProbability`. Son constructeur prendra comme argument deux tableaux correspondant respectivement à celui des angles de déplacement envisagés (en radians, voir ci-dessous) et à P_m, celui des probabilités associées. Il faudra vérifier que les deux tableaux (ne sont pas `null` et) ont la même taille. En revanche, il n'est pas requis de vérifier que la somme des probabilités fasse 1.

Si les données sont invalides, le constructeur lancera une `IllegalArgumentException`.

Notes :

1. Comme les tableaux sont manipulés par le biais de références en Java, il faudra copier leur contenu. La méthode Java `clone` peut ici être utilisée car les entrées des tableaux fournis sont de type primitif.
2. La classe fournie `Utils` (dans le répertoire `utils/`) met à disposition quelques méthodes utiles pour réaliser certains contrôles usuels, typiquement la non `null`lité des tableaux.

[**Fin de notes**]

Dotez également la classe `RotationProbability` de deux méthodes

```
double[] getAngles()
```

et

```
double[] getProbabilities()
```

qui retourneront respectivement (des copies de) le tableau d'angles de déplacement envisagés et le tableau de probabilités associées.

Intégration à `Animal`

Vous pourrez alors doter `Animal` d'une nouvelle méthode, protégée :

```
RotationProbability computeRotationProbs()
```

qui retourne une valeur du nouveau type défini ci-dessus.

La valeur retournée ne nécessitera aucun calcul. Elle encapsulera simplement les valeurs suivantes :

- pour les angles :

  ```
  ( -180, -100, -55, -25, -10, 0, 10, 25, 55, 100, 180 )
  ```

Les valeurs sont ici données en degrés pour une meilleure lisibilité, mais il faudra les convertir en radians avant de les passer à `RotationProbability` (en utilisant par exemple

```
double Math.toRadians(double angle)
```

- et pour les probabilités associées :

```
( 0.0000, 0.0000, 0.0005, 0.0010, 0.0050,
  0.9870,
  0.0050, 0.0010, 0.0005, 0.0000, 0.0000 )
```

Cette nouvelle méthode pourra librement être redéfinie par les sous-classes.

L'idée est maintenant de modifier la méthode `move()` de sorte à ce qu'avant de calculer la nouvelle position de l'animal, on lui fasse éventuellement subir une rotation (en s'aidant des données fournies par `computeRotationProbs`). Il vous est conseillé de créer une méthode privée pour ajouter la rotation (p.ex. `rotate()`)

Vous pouvez utiliser la méthode publique statique

```
pickValue(double[] values, double[] probs)
```

de la classe `Utils` en passant les angles pour `values` et les probabilités associées aux angles pour `probs`.

Pour un effet un peu plus réaliste, la tentative de rotation n'aura pas lieu à chaque pas de temps, mais à intervalle régulier toute de même.

Pour mettre en œuvre cela, vous pouvez ajouter à l'animal un compteur de type `Time` (initialisé à `Time.ZERO`), nommé `rotationDelay`, mesurant le temps écoulé depuis la précédente rotation. Vous ferez alors en sorte que l'animal ne fasse de tentative de rotation qu'au bout de l'écoulement du laps de temps `ANIMAL_NEXT_ROTATION_DELAY`. Vous devrez également prendre en compte le cas où le pas de `dt` est supérieur à `ANIMAL_NEXT_ROTATION_DELAY`. Vous pourrez vous inspirer de votre code écrit pour `FoodGenerator` (le principe est exactement le même !).

Voyons maintenant comment faire subir une éventuelle rotation à l'animal selon le modèle prévu.

L'idée est la suivante :

- on tire au hasard un angle de rotation gamma en utilisant les données liées aux probabilités de rotations (méthode `computeRotationProbs()`) ;
- et on ajoute gamma à l'angle de direction.

Tests et soumission

Tests locaux

Nous vous encourageons, comme toujours à continuer d'ajouter vos propres tests ponctuels dans le fichier `ch/epfl/moocprog/tests/Main.java`. Pour certains aspects du programme c'est un peu plus difficile maintenant en raison de l'aléatoire, mais vous pouvez vous assurer assez simplement du bon fonctionnement de certains composants de base (comme le résultat des méthodes de la classe `RotationProbability` par exemple).

Pour tester globalement le comportement de tout le système et de vos ajouts récents, vous êtes bien sûr encouragés à utiliser aussi l'interface graphique.

Si vous lancez ce programme avec les valeurs d'origine du fichier `res/app.cfg` vous devriez voir assez peu de changements dans le mode de déplacement de vos termites. En augmentant la fréquence des rotations (`ANIMAL_NEXT_ROTATION_DELAY` à 0.001 par exemple), vous devriez constater que vos termites ont des petits changements de direction aléatoires et se déplacent de façon plus réaliste en privilégiant toutefois les directions qui leur font face (elles ne pivotent pas abruptement sur elles-mêmes).

Une petite vidéo d'exemple de simulation obtenue à ce stade est disponible dans le matériel de la semaine 3.

Soumission

Pour soumettre votre devoir au correcteur automatique, il faut créer un fichier ZIP contenant **tous** vos fichiers sources personnels, depuis la racine `ch/` ; ceux de cette étape, mais aussi ceux de l'étape précédente. Concrètement, pour ce devoir ci, votre fichier ZIP doit donc contenir exactement les fichiers suivants :

```
ch/epfl/moocprog/AnimalEnvironmentView.java
ch/epfl/moocprog/Animal.java
ch/epfl/moocprog/Anthill.java
ch/epfl/moocprog/Ant.java
ch/epfl/moocprog/AntSoldier.java
ch/epfl/moocprog/AntWorker.java
ch/epfl/moocprog/Environment.java
ch/epfl/moocprog/FoodGeneratorEnvironmentView.java
ch/epfl/moocprog/FoodGenerator.java
ch/epfl/moocprog/Food.java
ch/epfl/moocprog/Pheromone.java
ch/epfl/moocprog/Positionable.java
ch/epfl/moocprog/RotationProbability.java
ch/epfl/moocprog/Termite.java
```

`ch/epfl/moocprog/ToricPosition.java`

Note : si c'est plus pratique pour vous, vous *pouvez* aussi faire un zip qui contient tout le sous-répertoire `ch/epfl/moocprog`, y compris, donc, les fichiers que nous vous avons fournis. Ces fichiers seront simplement ignorés par le correcteur automatique (et remplacés par les nôtres). L'énoncé « *Procédure de soumission* » de la semaine 1 vous indique comment procéder. **Avant de passer à l'étape suivante, n'oubliez pas de faire une sauvegarde de l'étape en cours**, comme indiqué dans le même document.